CHAPTER 5

# Advanced File Processing

**case** Dominion Consulting is evaluating its programming staff and the staff's current workload. Management wants a Programmer Activity Status Report that shows programmers' names and the number of projects that each programmer is working on. Your assignment: to design and create the files necessary to obtain the data and then produce the report.

LESSON A

objectives

**After studying this lesson, you should be able to:**

- **Use the pipe operator to redirect the output of one command to another command**
- **Use the grep command to search for a specified pattern in a file**
- **Use the uniq command to remove duplicate lines from a file**
- **Use the comm and diff commands to compare two files**
- **Use the wc command to count words, characters, and lines in a file**
- **Use the manipulate and format commands: sed, tr, and pr**

# Selecting, Manipulating, and Formatting Information

Creating the Programmer Activity Status Report challenges you to make practical use of your UNIX file processing skills. You used many file processing commands in previous chapters. This lesson introduces new commands that let you complete advanced file processing tasks. Lesson B focuses on how to design a new application. You learn to design the application, create its files and shell scripts, and produce the final report.

## Advancing Your File Processing Skills

In Chapter 4 you learned to use several UNIX commands to extract and organize information from existing files and transform that information into a useful format. This chapter explains how to use other file processing commands, which are organized into two categories: select commands, and manipulation and transformation commands. Table 5-1 lists the select commands, which extract information.

| Command | Purpose |
|---|---|
| comm | Compare sorted files and show differences |
| cut | Select columns (fields) |
| diff | Compare and select differences in two files |
| grep | Select lines or rows |
| head | Select header lines |
| tail | Select trailing lines |
| uniq | Select unique lines or rows |
| wc | Count characters, words, or lines in a file |

**Table 5-1:** Select commands

The **manipulation and transformation commands** alter and transform extracted information into useful and appealing formats. Table 5-2 lists these commands.

| Command | Purpose |
|---|---|
| awk | Invoke Awk, a pattern-scanning and processing language |
| cat | Concatenate files |
| chmod | Change security mode of a file or directory |
| join | Join two files, matching row by row |
| paste | Paste multiple files, column by column |
| pr | Format and print |
| sort | Sort and merge multiple files |
| sed | Edit data streams |
| tr | Translate character by character |

**Table 5-2:** Manipulation and transformation commands

# Using the Select Commands

You used the head and tail commands in Chapter 1, and the cut command in Chapter 4. Now you can work with the grep, diff, uniq, comm, and wc commands, which also let you process files.

> **Note: The command usage in this chapter demonstrates how commands generally work. Appendix B, "Syntax Guide to UNIX Commands," more completely describes these commands.**

## Using Pipes

As you have seen, most UNIX commands take their input from stdin (the standard input device) and send their output to stdout (the standard output device). You have also used the > operator to redirect a command's output from the screen to a file, and the < operator to redirect a command's input from the keyboard to a file. The pipe operator (|) redirects the output of one command to the input of another command. The pipe operator is used in the following manner:

```
first_command | second_command
```

The pipe operator connects the output of the first command with the input of the second command. For example, the output of the ls command is commonly redirected to the more command. This technique allows you to scroll through a directory listing that does not fit on a single screen.

**To redirect the output of the ls command to the more command:**

1. Type **ls –l /etc** and press **Enter**. The output of the command scrolls quickly by.
2. Type **ls –l /etc | more**.

   Notice the output fills the screen and pauses with the prompt "—More—" displayed on the bottom line. Each time you press the spacebar, the output advances to the next screen. Press the spacebar until the command has finished.

The pipe operator can connect several commands on the same command line, in the following manner:

```
first_command | second_command | third_command . . .
```

**To connect several commands with the pipe operator:**

- Type **ls /etc | sort –r | more** and press **Enter**. This command redirects the directory listing of the /etc directory to the sort –r command. sort –r sorts the directory listing in reverse order. The sort command's output is redirected to the more command.

  You see the directory listing of /etc in reverse order, one screen at a time.

## Using the grep Command

Use the grep command to search for a specified pattern in a file, such as a particular word or phrase. UNIX finds and then displays the line containing the pattern you specified. As you recall from Chapter 1, you can use the head command to retrieve the first 10 lines of a file. You can combine the grep and head commands to retrieve only the first 10 lines containing the word or phrase. For example, use grep with head to find the first 10 lines in /etc/termcap that contain the characters "IBM."

**To display lines in a file containing a particular word or phrase:**

1. To see all the lines in the /etc/termcap file that contain the characters "IBM," type **grep IBM /etc/termcap** and press **Enter**. There are numerous lines and the output scrolls by quickly.
2. Redirect the output of the grep command to the input of the more command. Type **grep IBM /etc/termcap | more** and press **Enter**.
3. Press the **spacebar** until the command is finished.
4. Redirect the output of the grep command to the head command. Type **grep IBM /etc/termcap | head** and press **Enter**.

   This means "look for 'IBM' in the /etc/termcap file, and display the first 10 lines you find."

The grep command's options and wildcard support allow powerful search operations.

**To expand the grep command's search capabilities with its options and wildcard support:**

1. To see each line in the /etc/termcap file that contains the word "Linux," type **grep Linux /etc/termcap** and press **Enter**. (Make sure to capitalize the L in Linux.)
2. Some lines in the file contain the word "linux" (with lowercase l). The search you performed in Step 1 only displayed the lines that contain "Linux." The –i option tells grep to ignore the case of the search characters. Type **grep –i linux/etc/termcap** and press **Enter**. You see the lines that contain either "Linux" or "linux."
3. The grep command supports wildcard characters in the search string. To see all the lines of the /etc/termcap file that start with "lin" followed by any other characters, type **grep –i ^lin/etc/termcap** and press **Enter**.
4. The grep command also supports wildcards in the filename. Type **grep linux /etc/*** and press **Enter**. You see the lines that contain "linux" from all the files in the /etc directory.

**5** The –l (lowercase L) option instructs grep to display only the names of the files that contain the search string. Type **grep –l linux /etc/*** and press **Enter**. You see the names of the files in the /etc directory that contain "linux."

The grep command also searches files for phrases that contain spaces, as long as the phrase is specified on the command line inside quotation marks. For example, grep can search for the phrase "IBM PC," as demonstrated in the next set of steps.

**To search a file for a phrase:**

- Type **grep "IBM PC" /etc/termcap** and press **Enter**.

  You see all lines in the /etc/termcap file that contain the phrase "IBM PC."

In the previous examples, grep searches the file whose name is specified on the command line. grep can also take its input from another command, through the pipe operator.

**To redirect the output of a command to the grep comand:**

- Type **ls /etc | grep magic** and press **Enter**. You see a list of the files whose names contain the word "magic."

**Note: The shell programming chapters that follow describe the grep file search command in great detail.**

## Using the uniq Command

The uniq command removes duplicate lines from a file. Because it compares only consecutive lines, the uniq command requires sorted input. The syntax of the uniq command is:

Syntax **uniq** [options] *file1 file2*

In its simplest form, the uniq command removes identical lines or rows from a file. The following command creates the file inventory. It contains all the lines in the parts file, with duplicate lines eliminated.

```
uniq parts > inventory
```

The –u option instructs uniq to output only the lines of the source file that are not duplicated. (If a line is repeated, it is not output at all.) Here is an example:

```
uniq –u parts > single_items
```

The –d option instructs uniq to output one copy of each line that has a duplicate. Unduplicated lines are not output. Here is an example:

```
uniq –d parts > multi_items
```

The next steps illustrate common uses of the uniq command. To practice the uniq command, start by using the cat command with the output redirection operator to create a new file, zoo1, in your working directory. The file lists animal names, food descriptions, pounds eaten daily, and food costs. The duplicate records in Step 1 are not mistakes, so type them as shown. Then you'll use the uniq command to remove duplicate records.

**To remove duplicate lines with the uniq command:**

1. After the $ command prompt, type the following text, pressing **Enter** at the end of each line:

   ```
   cat > zoo1
   Monkeys:Bananas:2000:850.00
   Lions:Raw Meat:4000:1245.50
   Lions:Raw Meat:4000:1245.50
   Camels:Vegetables:2300:564.75
   Elephants:Hay:120000:1105.75
   Elephants:Hay:120000:1105.75
   ```

2. Press **Ctrl+Z**.
3. To use uniq to remove duplicate lines from the zoo1 file and use the output redirection operator to create the new file zoo2, type **uniq zoo1 > zoo2** and press **Enter**.
4. To use the cat command to display the zoo2 file, type **cat zoo2** and press **Enter**.
5. You see the contents of zoo2 as listed next. Notice that the uniq command removed the duplicate lines.

   ```
   Monkeys:Bananas:2000:850.00
   Lions:Raw Meat:4000:1245.50
   Camels:Vegetables:2300:564.75
   Elephants:Hay:120000:1105.75
   ```

## Using the comm Command

Like the uniq command, the **comm** command identifies duplicate lines. Unlike the uniq command, it doesn't delete duplicates and it works with two files rather than one. The comm command locates identical lines within two identically sorted files. It compares lines common to file1 and file2 and produces three-column output:

- The first column contains lines found only in file1.
- The second column contains lines found only in file2.
- The third column contains lines found in both file1 and file2.

The syntax of comm is:

Syntax comm [options] *file1 file2*

To practice using the comm command, start by creating the file my_list. Duplicate the file and then use the comm command to compare the two files.

**To use the comm command to compare files:**

1. To create the file my_list, after the $ command prompt, type **cat > my_list** and press **Enter**.
2. Type the following text, pressing **Enter** at the end of each line:

```
Football
Basketball
Skates
```

3. Press **Ctrl+Z**.
4. To copy my_list to a second file, your_list, type **cp my_list your_list** and press **Enter**.
5. Now use the comm command to compare my_list to your_list. Type **comm my_list your_list** and press **Enter**.
6. You see the three-column output. Notice that the lines in the third column are those that both files contain. The files are identical.

```
Column 1        Column 2        Column 3
                                Football
                                Basketball
                                Skates
```

7. Now add a new line to my_list. Type **cat >> my_list** and press **Enter**.
8. Type **Books** and press **Enter**.
9. Press **Ctrl+Z**.
10. Use comm to compare my_list to your_list again. Type **comm my_list your_list** and press **Enter**.

**11** You see the three-column output, with the unique new line in my_list in column 1.

```
Column 1        Column 2        Column 3
                                Football
                                Basketball
                                Skates
Books
```

**To find differences between two files:**

**1** After the $ command prompt, type **diff zoo1 zoo2** and press **Enter**.

**2** You see this information:

```
3d2
< Lions:Raw Meat:4000:1245.50
5d3
< Elephants:Hay:120000:1105.75
```

This means that you need to delete the third and fifth lines from zoo1 so the file matches zoo2.

**3** To reverse the comparison order, type **diff zoo2 zoo1** and press **Enter**.

**4** You see this information:

```
2a3
> Lions:Raw Meat:4000:1245.50
3a5
> Elephants:Hay:120000:1105.75
```

This means that you need to add the two lines shown to zoo2, so the file matches zoo1.

## Using the diff Command

The diff command attempts to determine the minimal set of changes needed to convert file1 to file2. The command's output displays the line(s) that differ. The code 3d2 displayed above the line indicates that you need to delete the third line in file1, so file1 matches file2. The d means delete, the 3 means the third line from file1, and the 2 means that file1 and file2 will be the same up to but not including line 2. The code 2a3 indicates you need to add a line to file1, so file1 matches file2. The a means add a line or lines to file1. The 3 means line 3 is to be added from file2 to file1. The 2 indicates that the line must be added following line 2.

You can use the diff command to find the differences between zoo1 and zoo2.

### Using the wc Command

Use the wc command to count the number of lines (option –l), words (option –w), and bytes or characters (option –c) in text files. You may specify all three options in the command line, that is, -lwc. If you enter the command without options, you see counts of lines, words, and characters in that order. You can use the wc command to count the number of lines in a new file, counters.

**To create a file and count its lines:**

1. After the $ command prompt, type **cat > counters** and press **Enter**.
2. Type this text, pressing **Enter** at the end of each line:

```
Linux is a full featured UNIX clone.
Linux blends the best of BSD and Sys V.
```

3. Type **Ctrl+Z**.
4. To find the number of lines in counters, type **wc –l counters** and press **Enter**. UNIX reports that the file contains two lines.
5. To find the number of bytes in counters, type **wc –c counters** and press **Enter**. UNIX reports that the file contains 77 bytes.
6. To find the number of words in counters, type **wc –w counters** and press **Enter**. UNIX reports that the file contains 16 words.
7. To count words, characters, and lines in counters, type **wc –lwc counters** and press **Enter**. UNIX reports the counts for lines (2), bytes (77), and words (16).

## Using the Manipulate and Format Commands

In addition to the manipulate and format commands you used in Chapter 4, you can also use the sed, tr, and pr commands to edit and transform data's appearance before you display or print it.

### Introducing sed

When you want to make global changes to large files, you need a more powerful editor than interactive editors like vi and Emacs. Another UNIX editor, **sed,** is designed specifically for that purpose. The minimum requirements to run sed are an input file and a command that lets sed know what actions to apply to the file. sed commands have two general forms:

| | |
|---|---|
| **Syntax** | **sed** [options] *'command' file(s)*<br>**sed** [options] –f *scriptfile file(s)* |
| **Dissection** | ■ The first form lets you specify an editing command on the command line, surrounded by single quotes.<br>■ The second form lets you specify a script file containing sed commands. |

You can use sed to work with a new file, unix_stuff, to display only certain lines and to replace text.

**To use sed to manipulate a file:**

1. To create a new file, unix_stuff, in your working directory, type **cat > unix_stuff** and press **Enter**.
2. Type this text, pressing **Enter** at the end of each line:

```
Although UNIX supports other database systems,
UNIX has never abandoned the idea of working with
flat files. Flat files are those that are based on pure
text with standard ASCII codes. Flat files
can be read by any operating system.
```

3. Press **Ctrl+Z**.
4. To display only lines 3 and 4, type **sed –n 3,4p unix_stuff** and press **Enter**. (The n option prevents sed from displaying any lines except those specified with the p command.)

   This means "find lines numbered (-n) 3 and 4 in the file Unix_stuff and display them (p)."

   You see lines 3 and 4:

```
flat files. Flat files are those that are based on pure
text with standard ASCII codes. Flat files
```

5. In sed, you can place two commands on one line. If you want to delete lines 3 and 4 and then display the file, you must use sed's –e option to specify multiple commands on the same line.
6. To delete lines 3 and 4 from unix_stuff and display the results, type **sed –n –e 3,4d –e p unix_stuff** and press **Enter**.

   You see the text:

```
Although UNIX supports other database systems,
UNIX has never abandoned the idea of working with
can be read by any operating system.
```

   **Note: Lines 3 and 4 are not actually deleted from the file but simply filtered out so they are not displayed.**

7. To display only lines containing the word, "Flat," type **sed –n /Flat/p unix_stuff** and press **Enter**.

   You see the text:

```
flat files. Flat files are those that are based
on pure text with standard ASCII codes. Flat files
```

8. To replace all instances of the word "Flat" with "Text," type **sed –n s/Flat/Text/p unix_stuff** and press **Enter**. (The s command substitutes one string of characters for another.)

   You see the text:

```
flat files. Text files are those that are based
on pure text with standard ASCII codes. Text files
```

To append new lines in sed, you must use the a\ command. This command appends lines after the specified line number. Like all other sed commands, it operates on all lines in the file if you do not specify a line number.

Next you can create a new script file, more_stuff. Include the append command, a\, in the file with the lines to be appended to the file unix_stuff. You must terminate each line, except for the last line of the file being added, with a backslash character. In the next steps, the $ preceding the a\ symbol tells sed to append more_stuff to unix_stuff after the last line in unix_stuff; without $, sed repeatedly adds all the lines in more_stuff after each line in unix_stuff.

**To create a script file to append lines to another file:**

1. To create the script file more_stuff, type **cat > more_stuff** and press **Enter**.
2. Type this text, pressing **Enter** at the end of each line:

```
$a\
Informix and Oracle, two major relational database\
companies have installed their RDBMS packages on UNIX\
systems for many years.
```

3. Press **Ctrl+Z**.
4. To use the sed command to run the script file, type **sed –f more_stuff unix_stuff** and press **Enter**.

   You see this text:

```
Although UNIX supports other database systems,
UNIX has never abandoned the idea of working with
flat files. Flat files are those that are based
on pure text with standard ASCII codes. Flat files
can be read by any operating system.
Informix and Oracle, two major relational database
```

```
companies have installed their RDBMS packages on UNIX
systems for many years.
```

5. Use vi to create the file Stuff_replace. Insert the following sed commands into the file:

```
s/UNIX/Linux/
s/abandoned/given up/
s/standard/regular/
```

   The lines in the file instruct sed to replace all occurrences of "UNIX" with "Linux," "abandoned" with "given up," and "standard" with "regular."

6. Execute sed, with the script file you created in Step 5, on the unix_stuff2 file. Redirect sed's output to the file unix_stuff2. Type **sed –f stuff_replace unix_stuff > unix_stuff2** and press **Enter**.

7. Type **cat unix_stuff2** and press **Enter**. You see the file with the changes specified by the stuff_replace script file.

## Translating Characters Using the tr Command

The tr command copies data from the standard input to the standard output, substituting or deleting characters specified by options and patterns. The patterns are strings and the strings are sets of characters.

| | |
|---|---|
| **Syntax** | **tr** [options] *string1 string2* |
| **Dissection** | ■ In its simplest form, tr translates each character in *string1* into the character in the corresponding position in *string2*. The strings are "quoted" with either single or double quotes.<br>■ Two options used most frequently are –d (delete character) and –s (substitute character). |

A popular use of tr is converting lowercase characters to uppercase characters. You can translate the file counters from lowercase to uppercase characters by specifying [a–z] as the lowercase characters and [A–Z] as the uppercase characters.

**To translate lowercase characters to uppercase characters in the file counters:**

- After the $ command prompt, type **tr [a-z] [A-Z] < counters** and press **Enter**.

You see these lines:

```
LINUX IS A FULL FEATURED UNIX CLONE.
LINUX BLENDS THE BEST OF BSD AND SYS V.
```

You can also use the –d option with the tr command to delete input characters found in *string1* from the output. This is helpful when you need to remove an erroneous character from the file.

**To delete specified characters from the counters file:**

- To delete the characters "full" from the output, type **tr –d "full" < counters** and press **Enter**.

  You see this text:

  ```
  Linx is a eatred UNIX cone.
  Linx bends the best o BSD and Sys V.
  ```

  Notice that the command deleted all characters in "full"—every f, u and l—from the output.

The –s option of the tr command checks for sequences of a *string1* character repeated several consecutive times. When this happens, tr replaces the sequence of repeated characters with one occurrence of the corresponding character from *string2*. For example, use the –s option when you need to change a field delimiter in a flat file from one character to another. For example, in the file zoo2, use tr to replace the field delimiter ":" with a space character, " ". First, use cat to display the file.

**To replace characters in the file counters:**

1. After the $ command prompt, type **cat zoo2** and press **Enter**.

   You see this text:

   ```
   Monkeys:Bananas:2000:850.00
   Lions:Raw Meat:4000:1245.50
   Camels:Vegetables:2300:564.75
   Elephants:Hay:120000:1105.75
   ```

2. Type **tr –s ":" " " < zoo2** and press **Enter**.

   You see this text:

   ```
   Monkeys Bananas 2000 850.00
   Lions Raw Meat 4000 1245.50
   Camels Vegetables 2300 564.75
   Elephants Hay 120000 1105.75
   ```

## Using the pr Command to Format Your Output

The pr command prints the specified files on the standard output in paginated form. If you do not specify any files or you specify a filename of "-", pr reads the standard input.

By default, pr formats the specified files into single-column pages of 66 lines. Each page has a five-line header, which, by default, contains the current file's name, its last modification date, current page, and a five-line trailer consisting of blank lines.

| | |
|---|---|
| **Syntax** | **pr** [options] [*file*…] |
| **Dissection** | ■ The three most frequently used options are –h (header-format), which lets you customize your header line; -d, which double-spaces output; and –l n, which sets the number of lines per page. |

Use pr to format and print the unix_stuff file. Use the pipe operator (|) to send the output to more, so the output screen does not flash by.

**To format a file:**

1. After the $ command prompt, type **pr –h "UNIX Files & Databases" <unix_stuff** (|) **more**, and press **Enter**.

   You see this text:

```
99-11-22 02:35           UNIX Files & Databases
Page 1

Although UNIX supports other database systems,
UNIX has never abandoned the idea of working with
flat files. Flat files are those that are based
on pure text with standard ascii codes. Flat files
can be read by any operating system.
```

   Now you can type the same command but add the –l 23 option to limit the number of lines per page to 23. Because the standard number of lines on most monitors is 24, you do not need to pipe the output to hold the screen.

2. Type **pr –l 23 –h "UNIX Files & Databases" <unix_stuff**, and press **Enter**.

   You see this text:

```
99-11-22 02:35           UNIX Files & Databases
Page 1

Although UNIX supports other database systems,
UNIX has never abandoned the idea of working with
flat files. Flat files are those that are based
on pure text with standard ascii codes. Flat files
can be read by any operating system.
```

## COMMAND SUMMARY

**Chapter 5, Lesson A commands**

| Command | Purpose | Options covered in this chapter |
|---|---|---|
| comm | Compare and output lines common to two files | |
| diff | Compare two files and determine which lines differ | |
| grep | Select lines or rows | -i ignores case<br>-l lists only filenames |
| pr | Format a specified file | -d double-spaces the output<br>-h customizes the header line<br>-l*n* sets the number of lines per page |
| sed | Specify an editing command or a script file containing sed commands | -a \ appends text after a line<br>-d deletes specified text<br>-e specifies multiple commands on one line<br>-n indicates line numbers<br>-p displays lines<br>-s substitutes specified text |
| tr | Translate characters | -d deletes input characters found in *string1* from the output<br>-s checks for sequences of *string1* repeated consecutive times |
| uniq | Remove duplicate lines to create unique output | |
| wc | Count the number of lines, bytes, or words in a file | -c counts the number of bytes or characters<br>-l counts the number of lines<br>-w counts the number of words |

## SUMMARY

- The UNIX file processing commands can be organized into two categories: select commands and manipulation and transformation commands. Select commands extract information. Manipulation and transformation commands alter and transform extracted information into useful and appealing formats.

- The uniq command removes duplicate lines from the file. You must sort the file, because uniq compares only consecutive lines.
- The comm command compares lines common to file1 and file2 and produces three-column output that reports variances between the files.
- The diff command attempts to determine the minimal set of changes needed to convert file1 into file2.
- The tr command copies data read from the standard input to the standard output, substituting or deleting the characters specified by options and patterns.
- sed is a file editor designed to make global changes to large files. Minimum requirements to run sed are an input file and a command that tells sed what actions to apply to the file.
- The pr command prints the standard output in pages.

## REVIEW QUESTIONS

1. Use select commands to _____.
   a. highlight text in a file
   b. format output
   c. extract information from a file
   d. redirect output to a file
2. Use manipulation and transformation commands to _____.
   a. change extracted information into a useful and appealing format
   b. change the file type
   c. extract information from a file
   d. redirect input
3. Which of these is a select command?
   a. cut
   b. cat
   c. sed
   d. uniq
4. Which of these is a manipulation and transformation command?
   a. cut
   b. cat
   c. sed
   d. uniq
5. Which of these commands processes two files simultaneously?
   a. comm
   b. diff
   c. sed
   d. uniq

6. The pr command produces _____.
   a. output just as cat does
   b. output just as less does
   c. paginated output to a line printer
   d. paginated output to the standard output

7. Which of these commands replaces colons in a file with blanks?
   a. sed ":" " " < filename
   b. tr –s ":" " " filename
   c. awk –F: '{ print " " }' filename
   d. tr –s ":" " " < filename

8. Which of these commands deletes characters from a file?
   a. sed –d ":" " " < filename
   b. tr –d ":" <filename
   c. tr –d: filename
   d. tr –d ":" " " filename

9. In sed, use the _____ option to append new lines.
   a. /a
   b. $/a
   c. add
   d. a\

10. Which of these commands converts lowercase characters in a file to uppercase characters?
    a. tr [a-z] [A-Z] filename
    b. tr "[a-z]" "[A-Z]" < filename
    c. tr "[A-Z]" "[a-z]" < filename
    d. tr a-z A-Z filename

11. The pipe (|) operator is used to _____.
    a. redirect output to a file
    b. redirect output to the screen
    c. redirect input from a file
    d. redirect one command's output to another command's input

12. What does the command grep Bob* /home/jill/members do?
    a. searches all files in the current directory whose names start with Bob for lines containing the string/home/jill/members. It displays the lines.
    b. searches the file /home/jill/members for lines that contain Bob followed by any other characters. It displays the lines.
    c. stores all the lines in all files that start with Bob in the /home/jill/members file.
    d. none of the above.

13. Which command displays on the screen the names of all files in the current directory with "jimmy" in their name?
    a. ls | grep jimmy
    b. ls | sort jimmy
    c. ls | tr jimmy
    d. sed jimmy | ls

14. The wc command counts _____.
    a. words
    b. lines
    c. characters
    d. all of the above

## EXERCISES

1. Use either the vi editor or cat >> to add lines to the your_list file. Use the comm command to compare your_list to my_list.
2. Convert the my_list file to all uppercase letters, and then run the comm command.
3. Remove all colon field separators from the zoo2 file.

## DISCOVERY EXERCISES

1. Create a new file, their_list, and enter these lines in the file:

   Radio

   Camera

   Boat

   Use the sed command option to add these lines to the end of your_list:

   Television

   Computer

   Stereo
2. Use the tr command to remove all colons from the /etc/passwd file and display the results on the screen.
3. Replace the word "Radio" with "Canoe" in the their_list file, and send the display to the standard output.
4. Use the cat command to create the file animals, containing the following lines:

   ```
   Dog
   Cat
   Horse
   Frog
   ```

   Next, use the sed command to change the word "Cat" to "Kitten."
5. Write and run a sed script on the file you created in Discovery Exercise 4. The script should change "Dog" to "Puppy" and "Frog" to "Tadpole."

LESSON B

objectives

**After studying this lesson, you should be able to:**

- **Design a new file processing application**
- **Design and create files to implement the application**
- **Use awk to generate formatted output**
- **Use cut, sort, and join to organize and transform selected file information**
- **Develop customized shell scripts to extract and combine file data**
- **Test individual shell scripts, and combine all scripts into a final shell program**

# Using UNIX File Processing Tools to Create an Application

## Designing a New File Processing Application

The most important phase in developing a new application is creating a design. The design defines the information an application needs to produce. The design also defines how to organize this information into files, records, and fields, which are **logical structures**, because each represents a logical entity such as a payroll file, an employee pay record, or an employee social security field. Files consist of records and records consist of fields.

Now you're ready to create the Programmer Activity Status Report for Dominion Consulting. The report will show programmers' names and the number of projects that each programmer is working on. Start by designing and creating the files, including the records and fields, and then using advanced file processing commands to select, manipulate, and format information in the report.

### Designing Records

The first task in the design phase is to define the fields in the records. These definitions take the form of a **record layout** that identifies each field by name and data type (such as numeric or non-numeric). Design the file record to store only those fields relevant to the record's primary purpose. For example, you need two files for Dominion Consulting: one for programmer information and another for project information. Include a field for the programmer's name in the programmer file record and a field for the project description field in the project file record. Do not store a programmer's name in a project file, even though the programmer may be assigned to the project. Conversely, do not store project names in the programmer files.

Allocating only the space needed for the records' necessary fields keeps records brief and to the point. Short records, like short sentences, are easier to understand. Likewise, the simpler you make your application, the better it performs. However, make sure to include a field that uniquely identifies each record in the file. For example, the programmer file record includes a programmer number field to separate programmers who may have the same name.

**Note: The programmer number field in the programmer file record should be numeric. Numeric fields are preferable to non-numeric fields for uniquely identifying records, because the computer interprets numbers faster than non-numeric fields. The project record can use a non-numeric project code to uniquely identify each project record, because Dominion project codes contain letters and numbers (EA-100).**

## Linking Files

Multiple files are joined by a **link**—a common field that each of the linked files share. Another important task in the design phase is to plan a way to join files, if necessary. For example, the programmer-project application uses the programmer's number to link the programmer to the project file. Add the programmer's number field to the project record to link programmers to projects.

**Note: The flexibility to gather information from multiple files, comprised of simple, short records, is the essence of a relational database system. UNIX includes several file processing commands that provide some of this relational database flexibility. You will implement some of these commands in this lesson.**

Before you begin to create files for the application, review the record layouts for the programmer and project files illustrated in Figure 5-1.

**Programmer File – Record Layout**

| Field Name | Data Type | Example |
|---|---|---|
| programmer_number | Numeric | 101 |
| lname | Alpha | Johnson |
| fname | Alpha | John |
| midinit | Alpha | K |
| salary | Numeric | 39000 |

**Field Separator is a colon :**

**Sample Record:**

101:Johnson:John:K:39000

**Project File – Record Layout**

| Field Name | Data Type | Example |
|---|---|---|
| project_code | Alpha | EA-100 |
| project_status | Numeric | 1 (*See Note) |
| project_name | Alpha | Reservation Plus |
| programmer_number | Numeric | 110 |

**Field Separator is a colon :**

**Sample Record:**

EA-100:1:Reservation Plus:110

*Note: Project Status Codes 1=Unscheduled 2=Started 3=Completed 4=Cancelled

**Figure 5-1:** Programmer and project application record layouts

## Creating the Programmer and Project Files

Now that you have reviewed the basic elements of designing and linking records, you can begin to implement your application design. As you recall from Chapters 2 and 3, UNIX file processing predominantly uses flat files. Working with these files is easy, because you can create and manipulate them with text editors like vi and Emacs. The flowchart in Figure 5-2 provides an overview and analysis of programmer project assignments as derived from the programmer and project files.

```
Filename: programmer                    Filename: project
```

```
101:Johnson:John:K:39000
102:King:Mary:K:39800
103:Brown:Gretchen:K:35000
104:Adams:Betty:C:42000
105:Utley:Amos:V:36000
106:Wilson:Patricia:B:39000
107:Culligan:Thomas:F:39000
108:Mitchell:Hillary:N:32800
109:Arbuckle:Margaret:F:46700
110:Ford:Terrence:H:40200
111:Greene:Sarah:L:41700
112:Rose:Richard:P:40200
113:Daniels:Allan:S:30500
114:Edwards:George:J:38500
```

```
EA-103:3:Personnel Evaluations:106
WE-206:1:Reservations:102
WE-207:4:Accounting - Basic:101
WE-208:2:Executive-Decision-Maker:102
NE-300:1:Region P & L:103
NE-302:1:Housekeeping Logs:104
NE-304:4:Maintenance Logs:105
```

Flowchart Logic

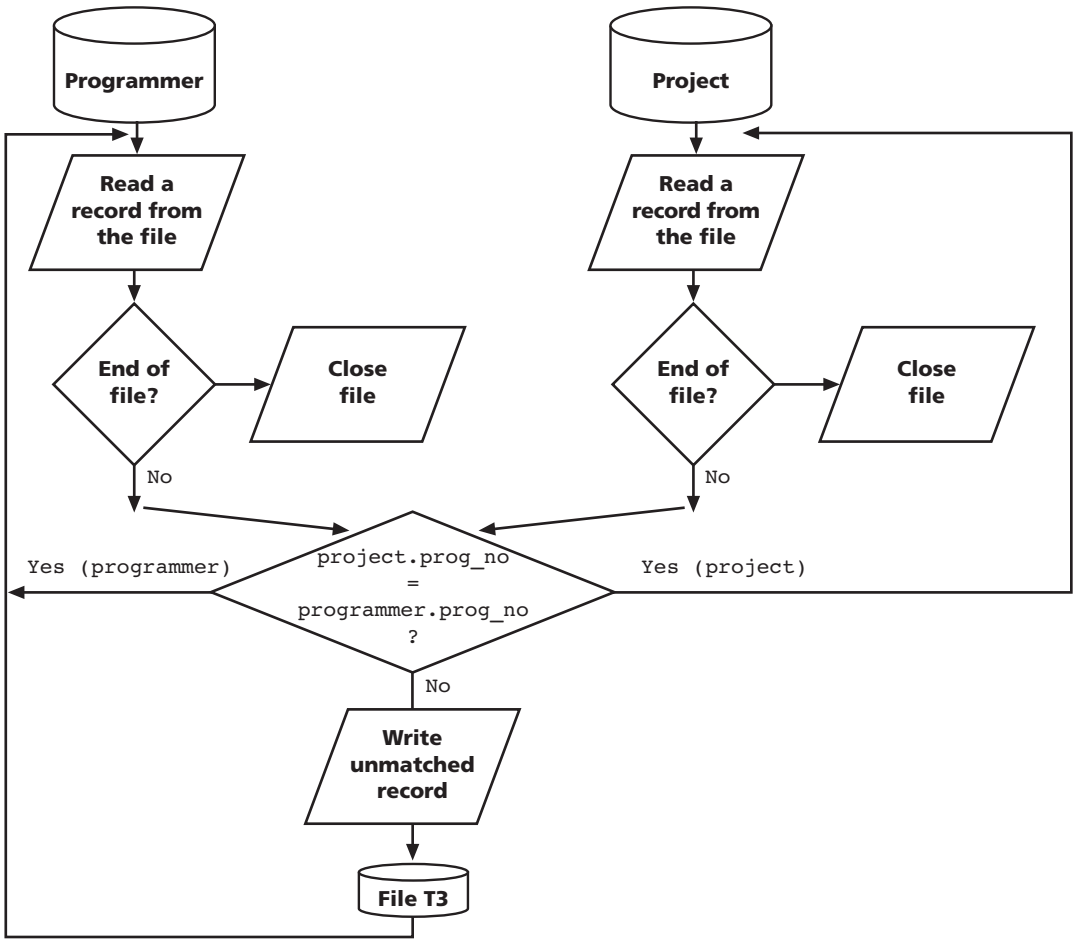


**Figure 5-2:** Overview and analysis of programmer assignments

Start by creating the programmer file in the vi editor.

**To create the programmer file:**

1. After the $ command prompt, type **vi programmer** and press **Enter**.
2. Type **a** to switch to insert mode, and then type the following text, pressing **Enter** at the end of each line except for the last line:

```
101:Johnson:John:K:39000
102:King:Mary:K:39800
103:Brown:Gretchen:K:35000
104:Adams:Betty:C:42000
105:Utley:Amos:V:36000
106:Wilson:Patricia:B:39000
107:Culligan:Thomas:F:39000
108:Mitchell:Hillary:N:32800
109:Arbuckle:Margaret:F:46700
110:Ford:Terrence:H:44700
111:Greene:Sarah:L:41700
112:Rose:Richard:P:40200
113:Daniels:Allan:S:30500
114:Edwards:George:J:38500
```

3. Press **Esc** to switch to command mode.
4. Type **:wq** to write the file and exit from vi.

**To create the project file:**

1. After the $ command prompt, type **vi project** and press **Enter**.
2. Type **a** to switch to insert mode, and type the following text, pressing **Enter** at the end of each line except for the last line:

```
EA-100:1:Reservation Plus:110
EA-100:1:Reservation Plus:103
EA-100:1:Reservation Plus:107
EA-100:1:Reservation Plus:109
EA-101:2:Accounting-Revenues Version 4:105
EA-101:2:Accounting-Revenues Version 4:112
EA-102:4:Purchasing System:110
EA-103:3:Personnel Evaluations:106
WE-206:1:Reservations:102
WE-207:4:Accounting - Basic:101
WE-208:2:Executive-Decision-Maker:102
NE-300:1:Region P & L:103
NE-302:1:Housekeeping Logs:104
NE-304:4:Maintenance Logs:105
```

3. Press **Esc** to switch to command mode.
4. Type **:wq** to write the file and exit from vi.

## Formatting Output

Chapter 4 introduced the awk command, which simplifies preparation of formatted output. Awk is actually a full-featured programming language and requires a chapter unto itself. The limited presentation here explains only the use of the printf action within the awk command, which formats output. The printf function has the syntax:

| | |
|---|---|
| **Syntax** | **printf** (*format, expr1, expr2, expr3 …*) |
| **Dissection** | ■ ***format*** is always required. It is an expression whose string value contains literal text and specifications of how to format expressions in the argument list. Each specification begins with a percentage character (%), which identifies the code that follows as a modifier (- to left justify; width to set size; .prec to set maximum string width or digits to the right of decimal point; s for an array of characters (string); d for a decimal integer; f for a floating-point decimal number). Enclosed in double quotes (" "), format is often referred to as a "mask" that overlays the data fields (***expr's***) going into it. |
| | ■ $expr1, $expr2, $expr3 are awk expressions that represent data fields. These expressions typically take the form $1, $2, $3, etc. In the programmer file, the expression $1 indicates the programmer number (the first field), $2 indicates the programmer's last name (the second field), and $3 indicates the programmer's first name (the third field). |

You can use the awk command and printf function to print the programmer_number, programmer last name, and programmer first name, all left-justified.

**To print three fields:**

■ After the $ command prompt, type **awk –F: '{printf "%d %-12.12s %-10.10s\n",$1,$2,$3 }' programmer** and press **Enter.**

Each % symbol in the format string corresponds with a $ field: The %d specifies how to display the $1 field, the %-12.12s specifies how to display the $2 field, and the %-10.10s specifies how to display the $3 field. Here is a breakdown of each specifier:

■ %d indicates that field $1 is to appear in decimal digits.

- %-12.12s indicates that field $2 is to appear as a string. The minus sign (-) specifies the string is to be left-justified. The 12.12 indicates the string should appear in a field padded to 12 spaces, with a maximum size of 12 spaces.
- %-10.10s indicates that field $3 is to appear as a string. The minus sign (-) specifies the string is to be left-justified. The 10.10 indicates the string should appear in a field padded to 10 spaces, with a maximum size of 10 spaces.

The spaces that appear in the format string are printed exactly where they appear in relation to the fields (one is printed after $1, and another is printed after $2). The trailing \n tells awk to skip a line after displaying the three fields.

Awk provides a shortcut to other UNIX file processing commands when you need to extract and format data fields for output. For example, although it takes a few lines of code, you can use the cut, paste, and cat commands to extract and display the programmers' last names and salaries. Start by using the cut command to extract the last name (field 2) from the programmer file, and store the output in temp1. Next use the cut command to extract the salary (field 5) from the programmer file, and store the output in temp2. Then use the paste command to combine temp1 and temp2, and create the file Progsal. Finally use the more command to display the output. You can also accomplish the same task with one awk command.

**To extract and display information using cut, paste, and more:**

1. After the $ command prompt, type **cut –f2 –d: programmer > temp1** and press **Enter**.
2. Type **cut –f5 –d: programmer > temp2** and press **Enter**.
3. Type **paste temp1 temp2 > progsal** and press **Enter**.
4. To use the more command to display the output, type **more progsal** and press **Enter**.

   You see output similar to the following excerpt:

```
Johnson      39000
King         39800
Brown        35000
Adams        42000
Utley        36000
...
```

**To accomplish the same task with one awk command:**

- After the $ command prompt, type **awk –F: '{ printf "%-10.10s %7.0f\n", $2, $5 }' programmer** and press **Enter**.

  You see output similar to the following excerpt:

```
Johnson      39000
King         39800
Brown        35000
Adams        42000
Utley        36000
...
```

## Cutting and Sorting

Now that you've created the programmer information file, you can select the programmer_number fields stored in the project file. These fields identify programmers who are currently assigned to projects. Refer to Figure 5-1 as you work through the next task.

Start by cutting the programmer_number fields from the project file (field 4) and piping (|) the output to a sort to place any duplicate numbers together. Pipe the sorted output to the uniq file to remove any duplicate programmer_numbers. Finally, redirect the output to a temporary file, t1. (The t1 file is a list of programmer numbers that identifies programmers who are assigned to projects.)

**To select fields from the project file:**

1. After the $ command prompt, type **cut –d: -f4 project | sort | uniq > t1** and press **Enter**.
2. To display the contents of t1, type **cat t1** and press **Enter**.

   You see the list of programmer numbers:

```
101
102
103
104
105
106
107
109
110
112
```

   The next step is to cut the programmer_number fields (field 1) from the programmer file and pipe the output as you did in Step 1. Call the new temporary file t2, which is a list of programmer numbers that identifies all programmers who work for Dominion.

3. Type **cut –d: -f1 programmer | sort | uniq > t2** and press **Enter**.
4. To display the contents of t2, type **cat t2** and press **Enter**.

   You see this list of programmer numbers:

```
101
102
103
104
105
106
107
108
109
110
111
112
113
114
```

   Now that t1 and t2 are sorted in the same order, you can match them. Use the comm command to select the lines from t1 that do not match lines in t2, and redirect the output to another file, t3, which lists programmer numbers of all programmers who are not assigned to projects.
5. Type **comm -13 t1 t2 > t3** and press **Enter**.
6. To display the programmer numbers for programmers who are not working on projects, type **cat t3** and press **Enter**.

   You see this list of programmer numbers:

```
108
111
113
114
```

   To display the names of unassigned programmers, you can now sort the programmer file in programmer_number order and write the output to t4.
7. Type **sort –t: +0 –1 –o t4 programmer** and press **Enter**.

   Now use the join command to match programmer_numbers in t4 and t3, and redirect the output to t5, which contains the names of all programmers who are not assigned to a project.
8. Type **join –t: -j1 1 –j2 1 –o 1.2 –o 1.3 –o 1.4 t4 t3 > t5** and press **Enter**.

9 To display the contents of t5, type **cat t5** and press **Enter**.

You see the following list of programmer names:

```
Mitchell:Hillary:N
Greene:Sarah:L
Daniels:Allan:S
Edwards:George:J
```

Now you can transform the output using the sed editor to eliminate the colon field separators in t5.

10 Type **sed –n 's/:/ /g'p < t5** and press **Enter**.

You see this list of programmer names:

```
Mitchell Hillary N
Greene Sarah L
Daniels Allan S
Edwards George J
```

## Using a Shell Program to Implement the Application

Your application for Dominion Consulting currently consists of many separate commands that must run in a certain order. As you recall from Chapter 4, you can create a script file to simplify the application. Store your commands in a script file, which in effect becomes a program. (When you develop an application, you should usually test and debug each command before you place it in your script file.) You can use the vi editor to create the script files.

**Note: Chapter 6 covers the subject of shell programming in detail.**

Shell programs should contain not only the commands to execute but also comments to identify and explain the program. Use the pound (#) character in script files to mark comments. This tells the shell that the words following # are a comment, not a UNIX command. The next steps show how to add comments to your shell programs. Start by using the vi editor to create the script file, pact. Notice that you begin by inserting comments to identify and explain the program.

**To create a script and add comments:**

1. After the $ command prompt, type **vi pact** and press **Enter**.
2. Type **a** to switch to insert mode, and then type the text below, pressing **Enter** at the end of each line:

```
# ========================================================
# Script Name:    pact
# By:             Instructor
# Date:           November 1999
# Purpose:        Create temporary file, pnum, to hold the
#                  count of the number of projects each
#                  programmer is working on. The pnum file
                   consists of
#                  prog_num and count fields
# ========================================================
cut -d: -f4 project | sort  |  uniq -c |
awk '{printf "%s:%s\n",$2, $1}' > pnum
# cut prog_num, pipe output to sort to remove duplicates
# and get count for prog/projects.
# output file with prog_number followed by count
```

3. Press **Esc** to switch to command mode.
4. Type **:wq** to write the file and exit from vi.

Now you can run pact and use the less command to display the contents of pnum.

**Note: The shell scripts in the following steps are executed using the sh command.**

**To run the script:**

1. After the $ command prompt, type **sh pact** and press **Enter**.
2. Type **less pnum** and press **Enter**.
3. You see these programmer numbers and count fields:

```
101:1
102:2
103:2
104:1
105:2
106:1
107:1
109:1
110:2
112:1
```

You now have a file that contains programmer numbers and the number of projects that each programmer is working on. Next, create a script file, pnumname, to extract the programmer names and numbers from the programmer file and redirect the output to the file pnn.

**To create another script file:**

1. After the $ command prompt, type **vi pnumname** and press **Enter**.
2. Type **a** to switch to insert mode, and then type the text below, pressing **Enter** at the end of each line:

```
# ================================================
# Script Name:   pnumname
# By:      JQD
# Date:    Nov 1999
# Purpose: Extract Programmer Numbers and Names
# ================================================
cut -d: -f1-4 programmer | sort -t: +0 -1 | uniq > pnn
# The above cuts out fields 1 through 4.
# The output is piped to a sort by programmer number.
# The sorted output is piped to uniq to remove duplicates.
# Uniq redirects the output to pnn.
```

3. Press **Esc** to switch to command mode.
4. Type **:wq** to write the file and exit from vi.
5. To run the shell program and use the less command to display the contents of pnn, type **sh pnumname** and press **Enter**.
6. Type **less pnn** and press **Enter**.
7. You see the programmer names and numbers, with duplicates eliminated:

```
101:Johnson:John:K
102:King:Mary:K
103:Brown:Gretchen:K
104:Adams:Betty:C
105:Utley:Amos:V
106:Wilson:Patricia:B
107:Culligan:Thomas:F
108:Mitchell:Hillary:N
109:Arbuckle:Margaret:F
110:Ford:Terrence:H
111:Greene:Sarah:L
112:Rose:Richard:P
113:Daniels:Allan:S
114:Edwards:George:J
```

Now you can create a script file, joinall, to join the files, pnn and pnumname, and redirect the output to pactrep.

**To create a script file that joins two files:**

1. After the $ command prompt, type **vi joinall** and press **Enter**.
2. Type **a** to switch to insert mode, and then type the text below, pressing **Enter** at the end of each line:

```
# ====================================
# Script Name:  joinall
# By:           JQD
# Date:        Nov 1999
# Purpose:Join pnum and pnn to create a report file
# ================================================
# join the files including the unassigned programmers
# You do this by placing the programmer names (pnn) file,
# first, in the join sequence.
join -t: -a1 -j1 1 -j2 1 pnn pnum  > pactrep
```

3. Press **Esc** to switch to command mode.
4. Type **:wq** to write the file and exit from vi.
5. To run joinall and use less to display the contents of pactrep, type **sh joinall** and press **Enter**.
6. Type **less pactrep** and press **Enter**.
7. You see the programmer names, including unassigned programmers' names:

```
101:Johnson:John:K:1
102:King:Mary:K:2
103:Brown:Gretchen:K:2
104:Adams:Betty:C:1
105:Utley:Amos:V:2
106:Wilson:Patricia:B:1
107:Culligan:Thomas:F:1
108:Mitchell:Hillary:N
109:Arbuckle:Margaret:F:1
110:Ford:Terrence:H:2
111:Greene:Sarah:L
112:Rose:Richard:P:1
113:Daniels:Allan:S
114:Edwards:George:J
```

## Putting It All Together to Produce the Report

An effective way to develop applications is to combine small scripts in a larger script file. Because you have already executed the individual scripts and tested for accuracy, you can now place each in a script file in the proper sequence to produce the final programmer and project report for Dominion.

Start by using the vi editor to create the shell script practivity. Use the :r command to retrieve the pact, pnumname, and joinall scripts and place them in the practivity shell script. You can then use vi's dd command to remove the lines indicated in the comments.

**To create a final shell script:**

1. After the $ command prompt, type **vi practivity** and press **Enter**.
2. Type **a** to switch to insert mode, and then type the text below, pressing **Enter** at the end of each line:

```
# ========================================================
# Script Name:  practivity
# By:           JQD
# Date:         Nov 1999
# Purpose:Generate Programmer Activity Status Report
# ========================================================
```

3. Press **Esc** to switch to command mode.
4. To retrieve the three script files, type **:r pact** and press **Enter**. Move the cursor to the end of the file, type **:r pnumname** and press **Enter**. Move the cursor to the end of the file, type **:r joinall** and press **Enter**.
5. Use the **dd** command to delete the comments from the script. For example, you could move the cursor to each line that begins with a #, and then type dd. You could also move the cursor to the first line beginning with a #, and then type 9dd to delete the current line and the eight comment lines after it. Do the same for the remaining comment lines in the file.

   Only these four lines should remain in the script:

```
cut -d: -f4 project | sort  |  uniq -c |
awk '{printf "%s:%s\n",$2, $1}' > pnum
cut -d: -f1-4 programmer | sort -t: +0 -1 | uniq > pnn
join -t: -a1 -j1 1 -j2 1 pnn pnum  > pactrep
```

**6** Type the following in the script:

```
# Print the report
awk '
BEGIN {
  { FS = ":"}
  { print "\tProgrammer Activity Status Report" }
  { "date" | getline d }
  { printf "\t   %s\n",d }
  { print "Prog#\t  *——Name———-*  Projects" }
  { print
"=========================================================" }
  }
  { printf "%-s\t%12.12s %12.12s %s\t%d\n",
           $1, $2, $3, $4, $5 } ' pactrep
# remove all the temporary files
   rm pnum pnn pactrep
```

**7** Press **Esc** to switch to command mode.

**8** Type **:wq** to write the file and exit from vi.

**9** After the $ command prompt, type **sh practivity** and press **Enter**.

You see the report:

```
     Programmer Activity Status Report
        Tue Nov 23 11:16:43 EST 1999
Prog#     *——Name———-*   Projects
=========================================================
101          Johnson      John K      1
102          King         Mary K      2
103          Brown        Gretchen K  2
104          Adams        Betty C     1
105          Utley        Amos V      2
106          Wilson       Patricia B  1
107          Culligan     Thomas F    1
108          Mitchell     Hillary N   0
109          Arbuckle     Margaret F  1
110          Ford         Terrence H  2
111          Greene       Sarah L     0
112          Rose         Richard P   1
113          Daniels      Allan S     0
114          Edwards      George J    0
```

## COMMAND SUMMARY

| Chapter 5, Lesson B commands | | |
|---|---|---|
| Command | Purpose | Options covered in this chapter |
| printf | Tell the Awk program what action to take | none |
| sh | Execute a shell script | none |

## SUMMARY

- The design of a file processing application reflects what the application needs to produce. The design also defines how to organize information into files, records, and fields, which are logical structures.
- Use a record layout to identify each field by name and data type (numeric or non-numeric). Design file records to store only those fields relevant to each record's primary purpose.
- Shell programs should contain commands to execute and comments to identify and explain the program. The pound # character used in script files denotes comments.
- Write shell scripts in stages so you can test each part before combining them into one script. Using small shell scripts and combining them in a final shell script file is an effective way to develop applications.

## REVIEW QUESTIONS

**1.** The first task of the design phase is to define the application's _____.
   a. records
   b. files
   c. shell programs
   d. logic

**2.** Used in script files, the ___ character denotes comments.
   a. *
   b. &
   c. ;
   d. #

3. What does %d indicate in an awk printf format string?
   a. Display the $ symbol.
   b. Display a field as a double-precision number.
   c. Display a field as a decimal digit.
   d. None of the above is correct.
4. What does %-10.10s indicate in an awk printf format string?
   a. Display a field as a left-justified string in a space padded to 10 characters, with a maximum of 10 characters displayed.
   b. Display a field as a right-justified string in a space padded to 10 characters, with a maximum of 10 characters displayed.
   c. Display a field as a single-precision number with 10 digits to the left and 10 digits to the right of the decimal point.
   d. None of the above is correct.
5. You can use the _____ command during a vi session to retrieve data from other files and insert it in a new vi file.
   a. :r
   b. :i
   c. :a
   d. :insert
6. Which of these awk commands outputs formatted fields?
   a. print
   b. display
   c. show
   d. printf

## EXERCISES

1. Write a one-line script file, lprog, using awk to print all fields in the programmer file. Use the sh command to execute lprog.
2. Write a one-line script file, lproj, using awk to print all fields in the project file. Use the sh command to execute lproj.
3. Use vi to create the script file lists, and then combine lprog and lproj into lists.

   Save lists and use the sh command to execute the lists script.

## DISCOVERY EXERCISES

1. Create the file called Software with the fields:
   - Project_number using the same numbers shown in the project file
   - Software Code using any three-digit number
   - Software Description such as Excel
2. Write a small application joining software records to matching records in the project file, and use the Awk program to print a report describing the software for each project.